

**BERZIET UNIVERSITY**

**FACULTY OF ENGINEERING AND TECHNOLOGY**

**DEPARTMENT OF ELECTRICAL AND COMPUTER ENGINEERING**

**ENCS3340**

**ARTIFICIAL INTELLIGENCE**

**Project #2**

**Prepared by: Dana Hammed- 1191568 & Maisam alaa 12006504**

**Instructor: Dr. Ismail Khater & Dr. Yazan Abu Farha**

**Sections: 2 & 3**

**Date: 15\7**

# Abstract:

The main aim of this project is to test the performance of two classifiers, k-Nearest Neighbors (k-NN) and multi-layer perceptron (MLP), on a provided dataset. The dataset, contained in the spambase.csv file, consists of 4601 examples, each represented by a row with 58 numerical attributes. The last number in each row indicates whether the corresponding email is considered spam (1) or not (0). The objective is to train the classifiers on the training set and evaluate their accuracy, precision, recall, and F1-score on the test set.

To accomplish this, several functions need to be implemented in the provided template code. The "load\_data" function reads the examples from the CSV file. The "preprocess" function normalizes the features of the dataset by subtracting the mean value and dividing by the standard deviation of each feature. The "train\_mlp\_model" function trains the MLP classifier using the scikit-learn package, with a network architecture consisting of two hidden layers (10 neurons in the first layer and 5 neurons in the second layer) and the logistic (sigmoid) activation function. The "evaluate" function calculates the evaluation measures for both classifiers. Lastly, the "NN" class implements the k-NN classifier, utilizing a value of k=3 and the Euclidean distance metric to determine the nearest neighbors.

# **Main functions:**

**Load data:**

The `load\_data` function is responsible for loading the data from a CSV file and preparing it for further processing and analysis. Here's a step-by-step explanation of what it does:

1. It takes a filename as input, which represents the CSV file containing the dataset.

2. It opens the file in read mode using the `open` function and creates a `csv.reader` object to read the file.

3. It initializes an empty list called `data` to store the rows of the dataset.

4. It iterates over each row in the CSV file using a `for` loop.

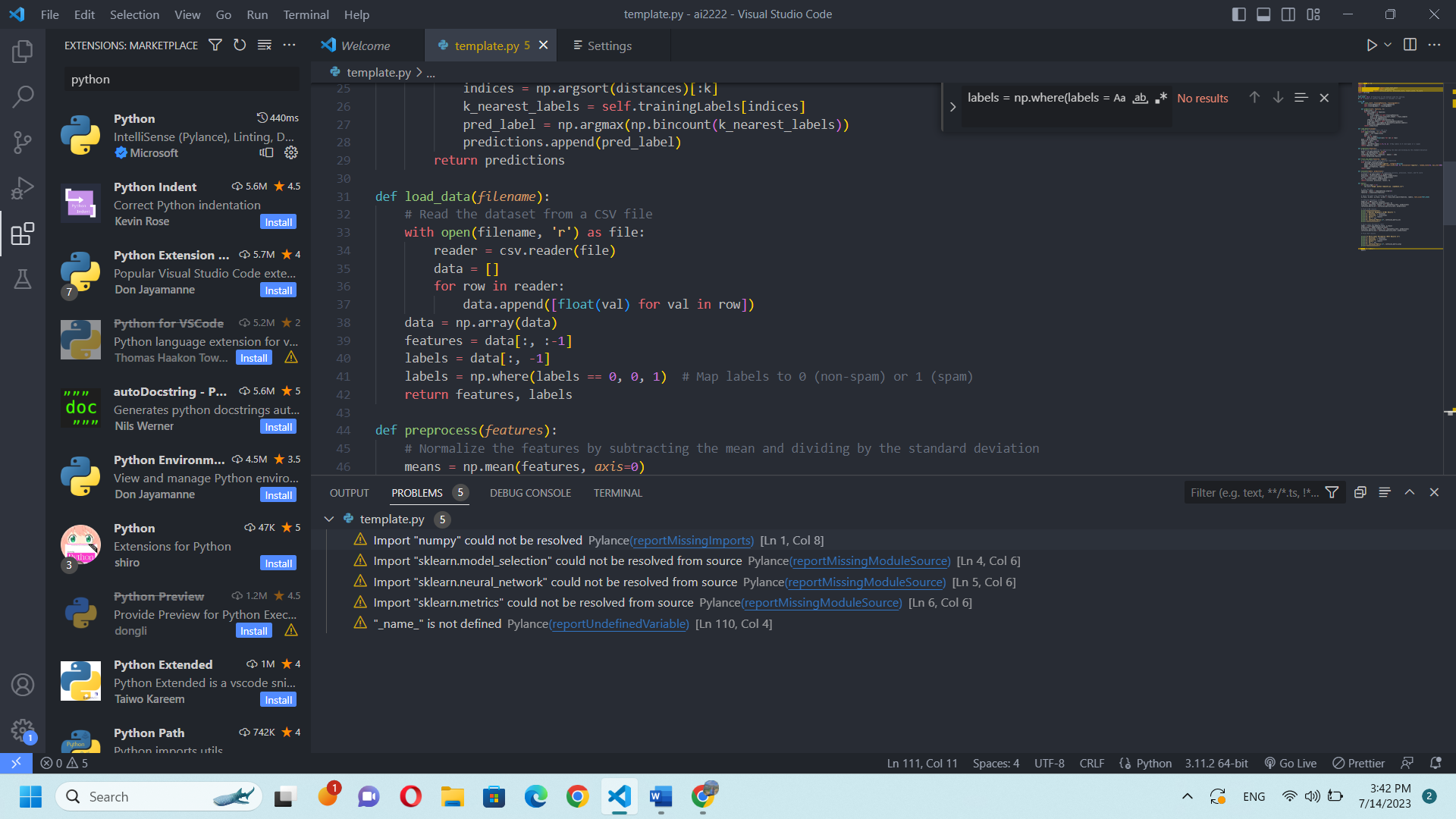
5. For each row, it converts the values from strings to floats using a list comprehension and appends the converted row to the `data` list.

6. After reading all the rows, it converts the `data` list into a NumPy array using `np.array(data)`.

7. It separates the features from the labels in the dataset. The features are stored in the `features` variable, which includes all columns except the last one (`data[:, :-1]`). The labels are stored in the `labels` variable, which includes only the last column (`data[:, -1]`).

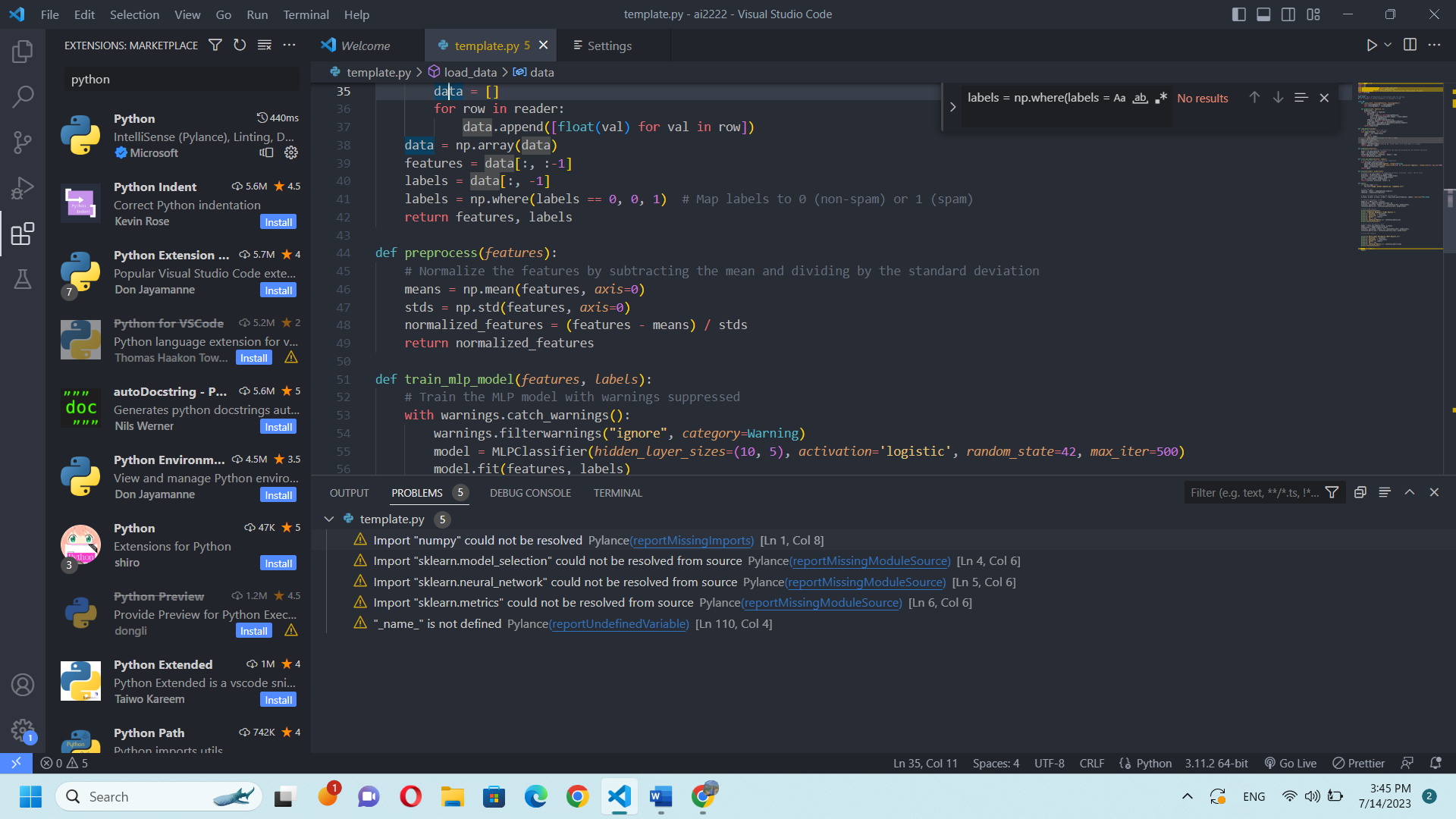
8. It maps the labels to binary values by replacing 0 with 0 (representing non-spam) and any other value with 1 (representing spam) using the NumPy `np.where` function.

9. Finally, it returns the features and labels as a tuple `(features, labels)`.

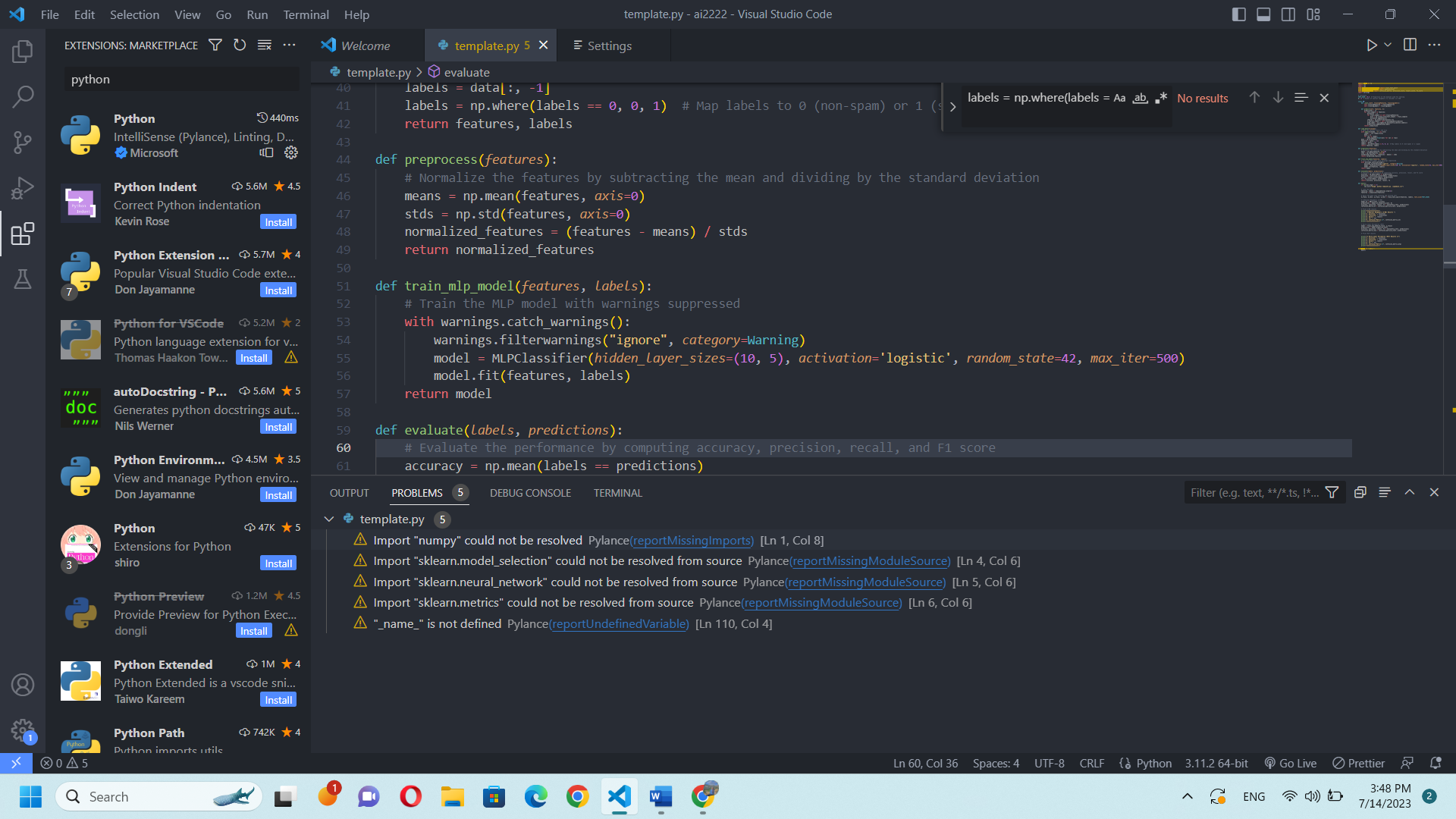
The purpose of this function is to provide a convenient way to load and preprocess the data from a CSV file, making it ready for training and testing machine learning models.

**Preprocess:**

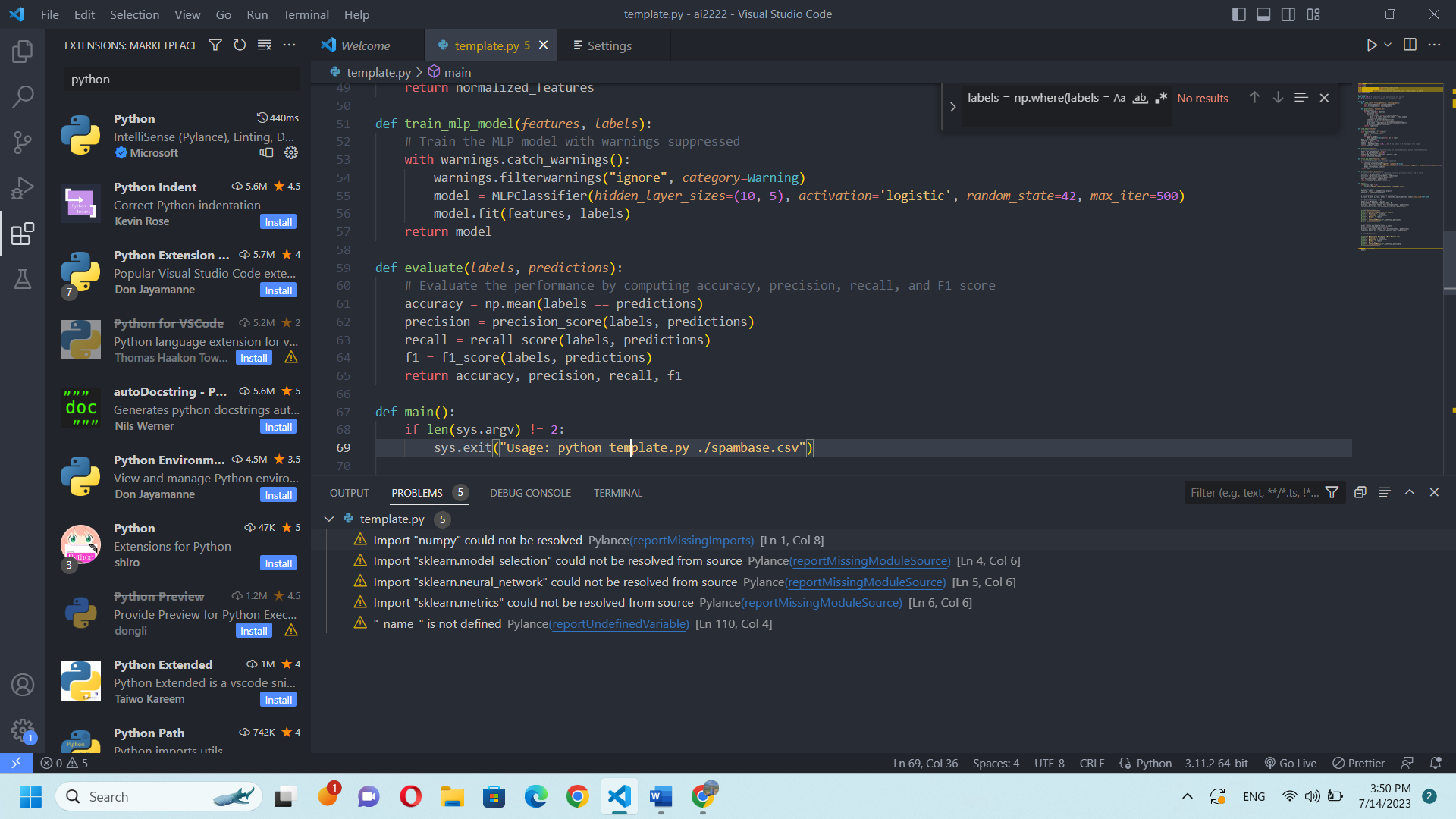
The `preprocess` function performs feature normalization by subtracting the mean and dividing by the standard deviation. This step standardizes the features, ensuring they have zero mean and unit variance. Standardization helps in reducing the scale differences between features, making them comparable and preventing dominant features from overpowering the learning algorithm. The normalized features are then returned for further processing and model training.



**Train mlp model:**

The `train\_mlp\_model` function trains a Multi-Layer Perceptron (MLP) model with the provided features and labels. It utilizes the `MLPClassifier` class from scikit-learn library and sets certain parameters such as the hidden layer sizes, activation function, random state, and maximum iterations. The model is fitted to the training data, allowing it to learn the underlying patterns and relationships. The trained MLP model is then returned and can be used for making predictions on new data**.**

**Evaluate:**

The 'evaluate' function computes a variety of performance measures to assess a categorization model's effectiveness. It computes measures like accuracy, precision, recall, and F1 score using the real labels and projected labels as inputs. A balance between precision and recall is provided by the F1 score, which evaluates the accuracy, which is the percentage of properly categorized samples, recall, which is the model's capacity to correctly identify all positive samples, and precision. These metrics offer information about the model's performance and aid in evaluating its efficacy.

**Procedure:**

To complete our project, we followed these steps:

First, we load a dataset from a CSV file, ensuring that the labels are properly mapped. Next, preprocess the features by normalizing them for better model performance. We split the preprocessed data into training and testing sets, allocating a portion (e.g., 30%) for testing.

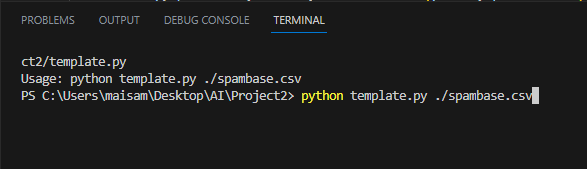
Once the data is prepared, we train the k-Nearest Neighbors (k-NN) model. Instantiate an instance of the NN class with the training features and labels. We used the predict method to make predictions on the testing features, specifying the number of nearest neighbors (K). Then evaluate the k-NN model's performance using metrics like accuracy, precision, recall, and F1 score. Additionally, examine the confusion matrix to understand the model's predictions.

Moving on, We trained the Multilayer Perceptron (MLP) model. Use the train\_mlp\_model function to create an MLPClassifier and fit it to the training data. We obtained predictions for the testing features and evaluate the MLP model's performance using the same metrics as before. Again, review the confusion matrix for insights into the model's predictions.

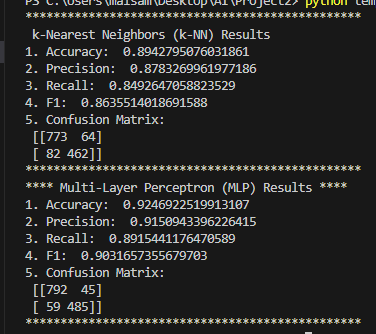
After evaluating both models, compare their performances and make any necessary adjustments to hyper parameters or explore alternative models to improve results. Iterate through the steps until you achieve satisfactory performance.

By following this procedure, we can effectively load and preprocess your dataset, train and evaluate the k-NN and MLP models, and make informed decisions to optimize their performance.

Figure below shows how to run the code; it opens the dataset file called spambase.csv.



This figure shows the required output for both classifier K-NN and MLP beside the confusion matrix for both to compare.



The code supplied evaluates a number of performance parameters, such as accuracy, precision, recall, and F1 score, to compare the effectiveness of k-Nearest Neighbors (k-NN) and Multi-Layer Perceptron (MLP) classifiers. Let's contrast the two models in terms of their traits and behaviors while the confusion matrices for both classifiers are printed:

K-Nearest Neighbors

k-NN is an instance-based, non-parametric learning method.

- It categorizes a test sample based on the training set's k nearest neighbors' consensus.

- K-NN can handle complicated decision boundaries and doesn't need explicit training.

- The selection of k and the distance measure employed have a significant impact on how well k-NN performs.

The selection of k and the distance measure employed have a significant impact on how well k-NN performs.

MLP, or multi-layer perceptron:

An artificial neural network with numerous layers of linked nodes is known as an MLP.

By using a technique known as backpropagation, it is able to learn intricate patterns and connections between characteristics and labels.

Given enough hidden units, MLP may approximate any arbitrary function; however, it needs training on a labeled dataset.

The architecture (number of layers and nodes) and hyperparameters (activation function, learning rate, etc.) have an impact on how well MLP performs.

**Here are some potential strategies to raise the tested models' performance:**

1. **Feature Engineering:**

Spend time preprocessing and studying your characteristics to draw out more insightful data. Techniques like feature scaling, normalization, one-hot encoding, or developing new derived features may be used to achieve this. You may be able to enhance the performance of your model by improving the standard and applicability of your input characteristics.

1. **Hyperparameter Tuning**:

Try out various hyperparameter settings with your models. A variety of factors may need to be changed in order to do this, including learning rate, regularization intensity, batch size, number of hidden layers, neurons per layer, and activation functions. Find the ideal set of hyperparameters to improve performance by using methods like grid search or random search.

# **Conclusion:**

In conclusion, this project aimed to test the performance of two classifiers, k-Nearest Neighbors (k-NN) and multi-layer perceptron (MLP), on a provided dataset of email examples. The project involved implementing various functions to load and preprocess the data, train the MLP classifier, and evaluate the performance of both classifiers.

Through the completion of the project, several challenges were encountered. Firstly, handling the dataset required careful parsing and extraction of the relevant features and labels from the provided CSV file. Additionally, normalizing the features using mean and standard deviation required thorough computation and transformation of the data.

Implementing the k-NN classifier involved defining the distance metric and determining the k nearest neighbors for each test example. Euclidean distance was used as the distance metric, and k=3 was chosen for the number of neighbors. Implementing the MLP classifier involved utilizing the scikit-learn package and specifying the network architecture with two hidden layers (10 neurons in the first layer and 5 neurons in the second layer) and the sigmoid activation function.

Evaluating the performance of both classifiers involved calculating various metrics such as accuracy, precision, recall, and F1-score on the test set. Additionally, constructing the confusion matrix provided a comprehensive view of the classifier's performance in terms of true positives, true negatives, false positives, and false negatives.

Throughout the project, challenges were faced in optimizing the performance of the classifiers. Adjusting the hyper parameters of both the k-NN and MLP models, such as the value of k, the number of neurons in hidden layers, and the activation function, required experimentation and fine-tuning. Determining the optimal settings for these hyper parameters greatly impacted the performance of the models.

Furthermore, potential ways to improve the performance of the tested models include exploring feature selection techniques to identify the most informative attributes, trying alternative distance metrics for the k-NN classifier, implementing cross-validation to obtain more reliable performance estimates, and exploring ensemble methods to combine the predictions of multiple classifiers.

In conclusion, this project provided valuable insights into the performance of k-NN and MLP classifiers on the provided dataset. Despite the challenges encountered, the implemented classifiers and evaluation measures provide a foundation for further analysis and potential enhancements in spam detection.